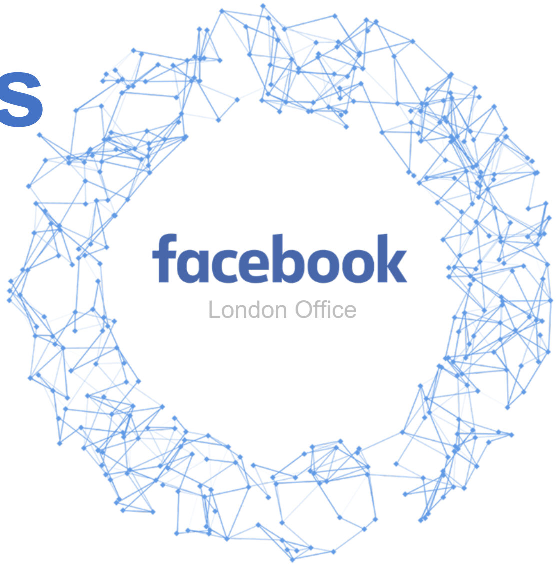
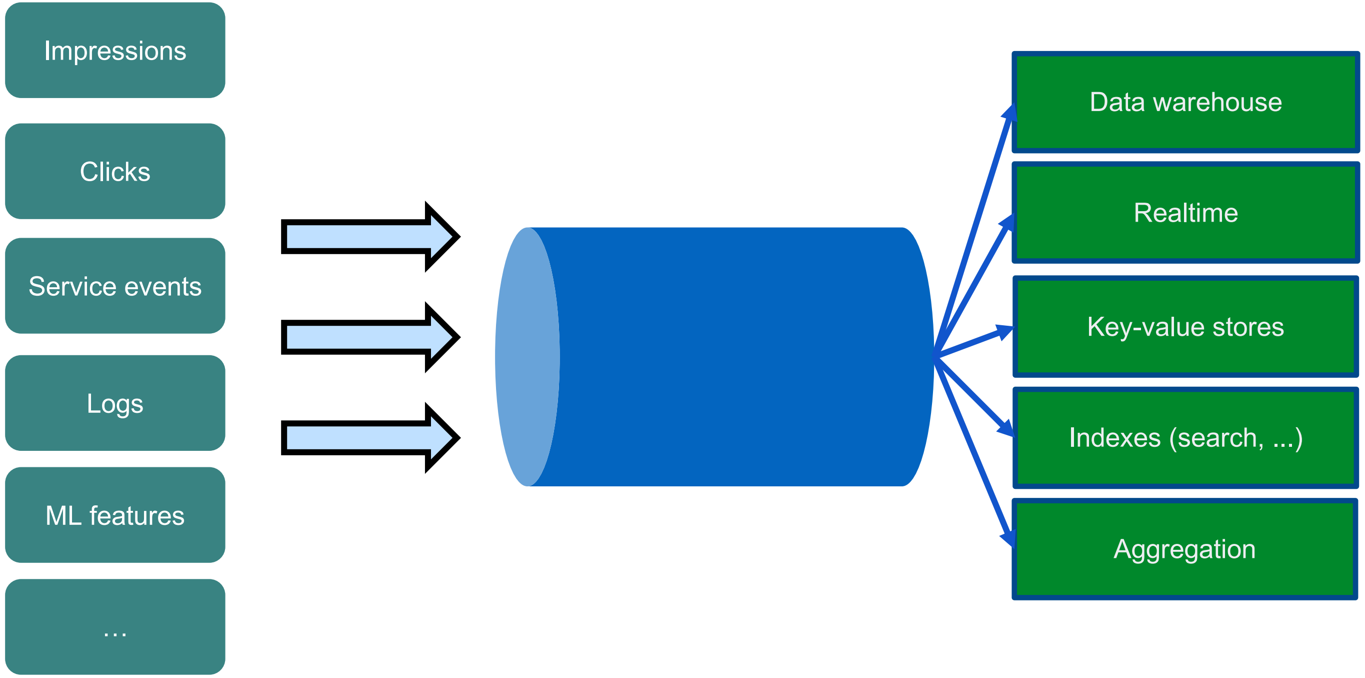
Architecture trade-offs in a planet-scale queueing system

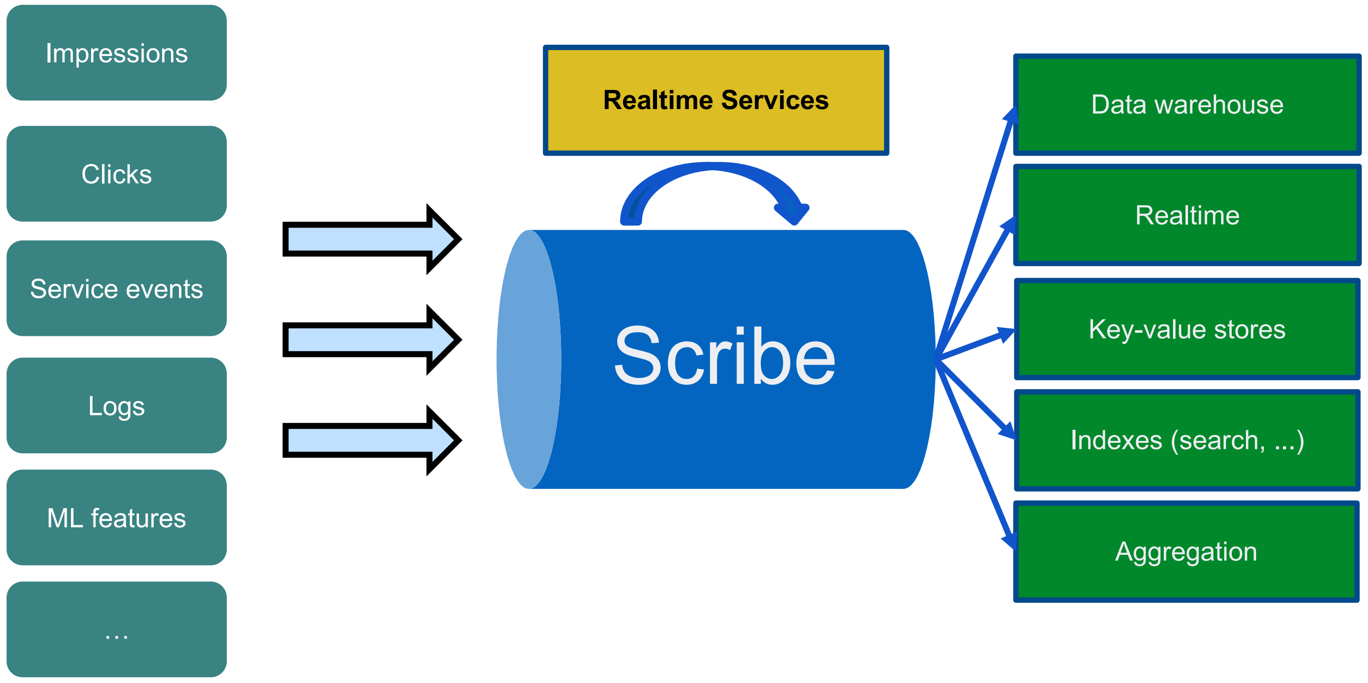
Manos Karpathiotakis



Getting data from point A to point B

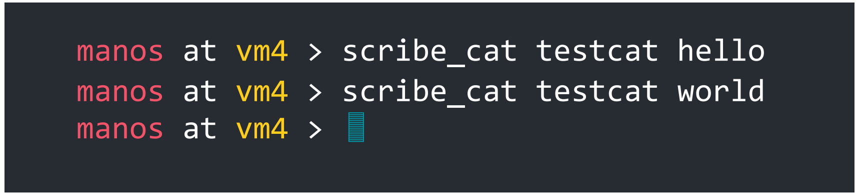


Getting data from point A to point B

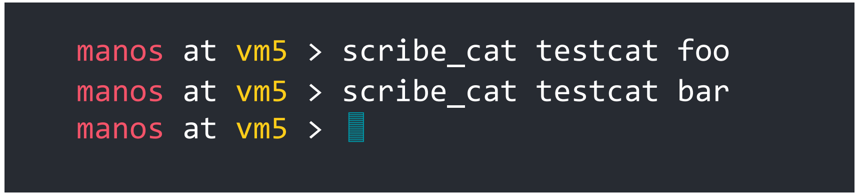


Distributed, buffered, multi-tenant pipe

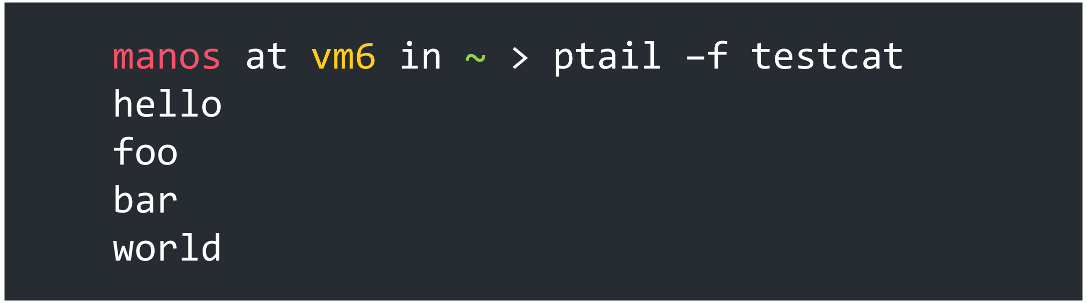
Hello World



```
manos at vm4 > scribe_cat testcat hello
manos at vm4 > scribe_cat testcat world
manos at vm4 >
```



```
manos at vm5 > scribe_cat testcat foo
manos at vm5 > scribe_cat testcat bar
manos at vm5 >
```



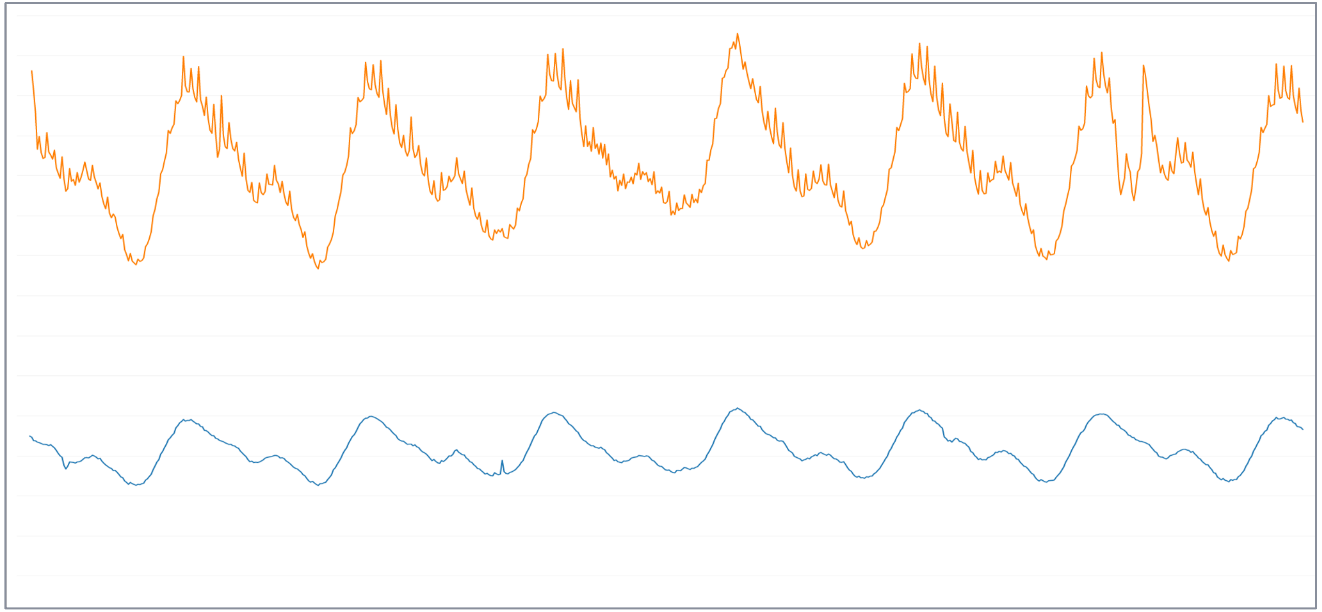
```
manos at vm6 in ~ > ptail -f testcat
hello
foo
bar
world
```

Logical stream abstraction

Scale

Read: 7TB/s

Write: 2.5TB/s



Millions of machines

Hundreds of thousands of categories/topics

Scale does not come for free

Retention in the days

Available on every machine

Latency seconds to minutes

Slightly lossy, 3 9s to 5 9s

Rough ordering guarantees

Retention in the days

Available on every machine

Latency seconds to minutes

Slightly lossy, 3 9s to 5 9s

Rough ordering guarantees

This talk

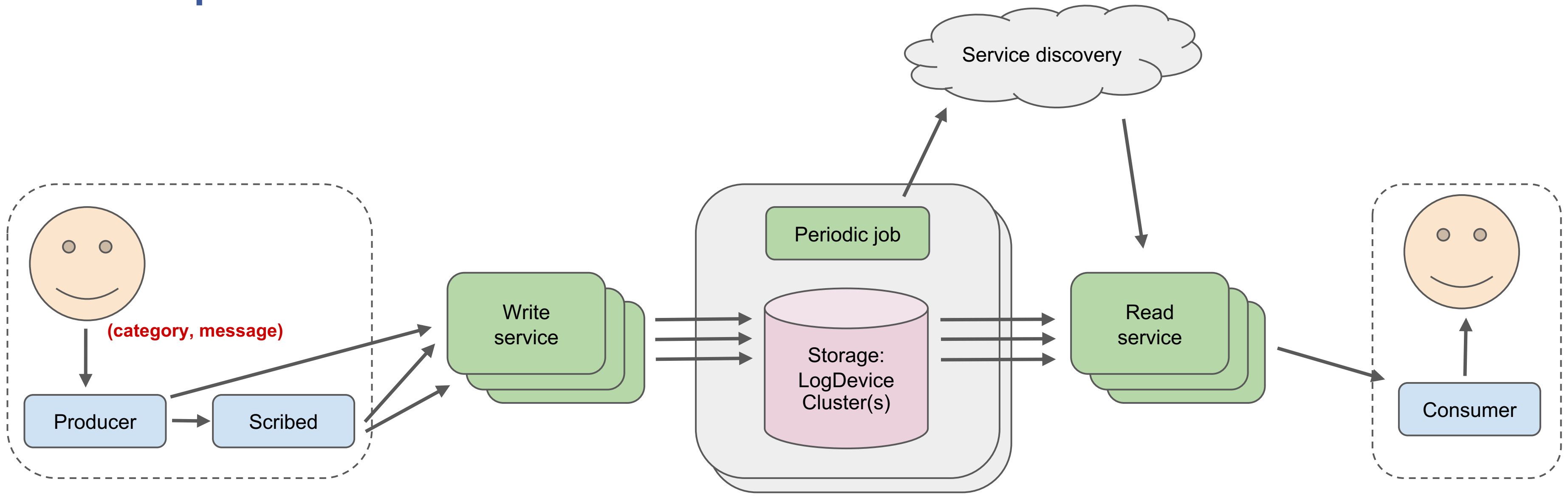
Data Completeness

Customers willing to lose data?!

- High volumes generated in real time (“nowhere to park the data”)
- Lossy or sampled upstream (“not making things worse”)
- Statistical in nature where small losses not critical (ML use cases)
- Data freshness imperative (“stale data is useless data”)

Multiple customers are unwilling to “pay” for completeness

Data plane



(Big) Aggregation Tree

Write path

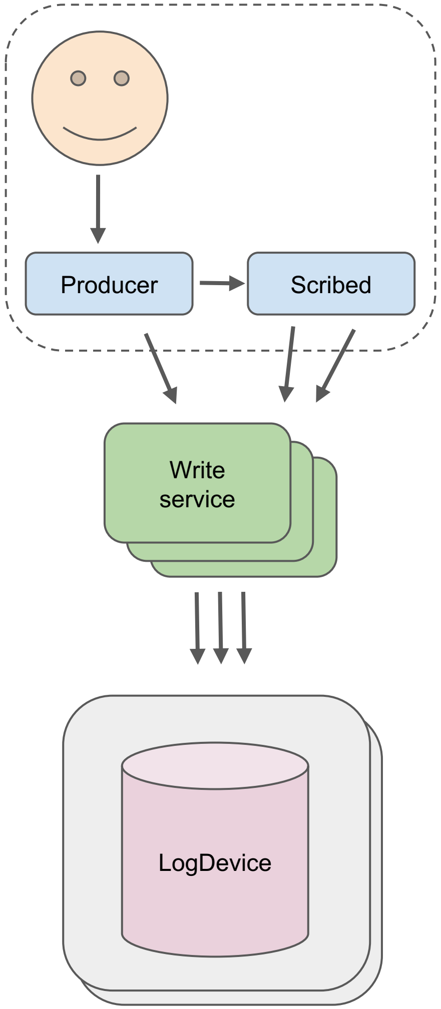
LogDevice is durable storage

But on the way, single copy in memory

When to acknowledge producer operations?

1. Once processed in the producer?
2. Once processed in the write service?
3. Once stored in LogDevice?

Offer customers multiple flavors



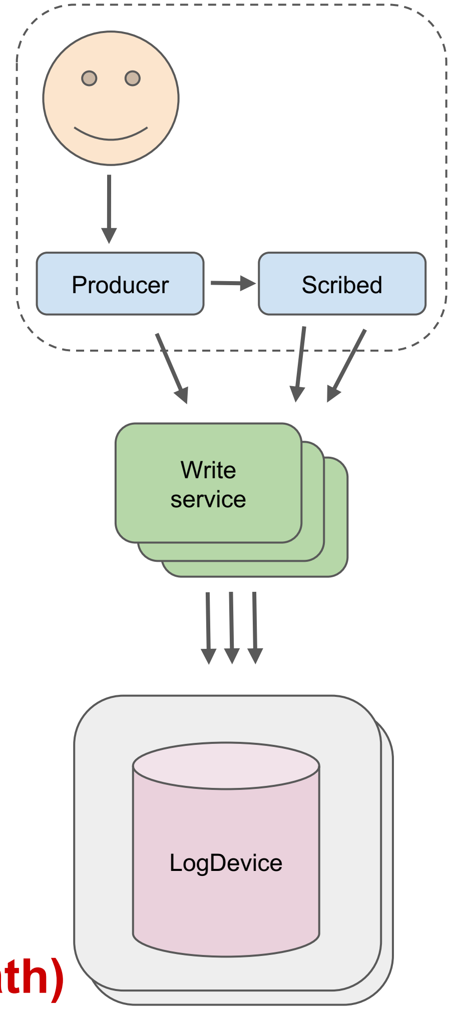
High durability flavor

Acknowledgement once stored in LogDevice

Increased duplication

Less aggressive batching => Lower throughput

Lower latency



At least once semantics (…on write path)

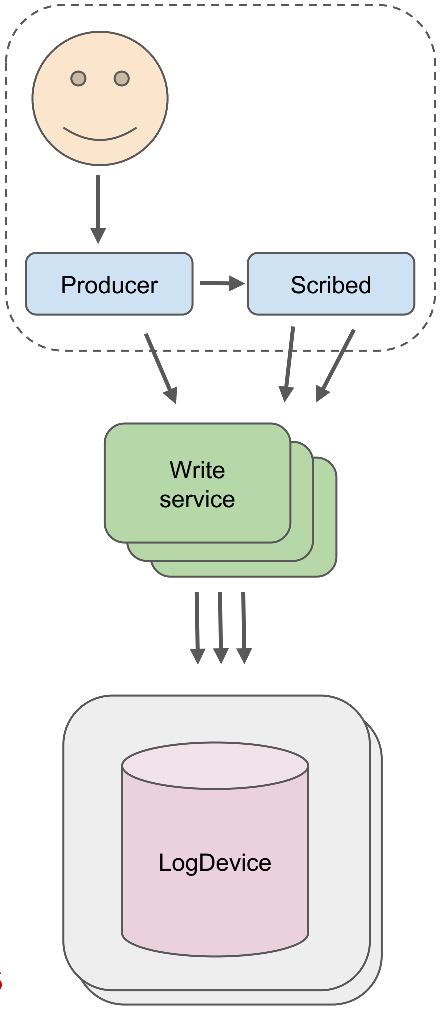
High throughput flavor

Producer (**optionally**) acknowledges upon receival

Accept small amount of data loss

Heavy batching provides high scalability

“Approximately once” semantics



“Approximately once” semantics

Read path

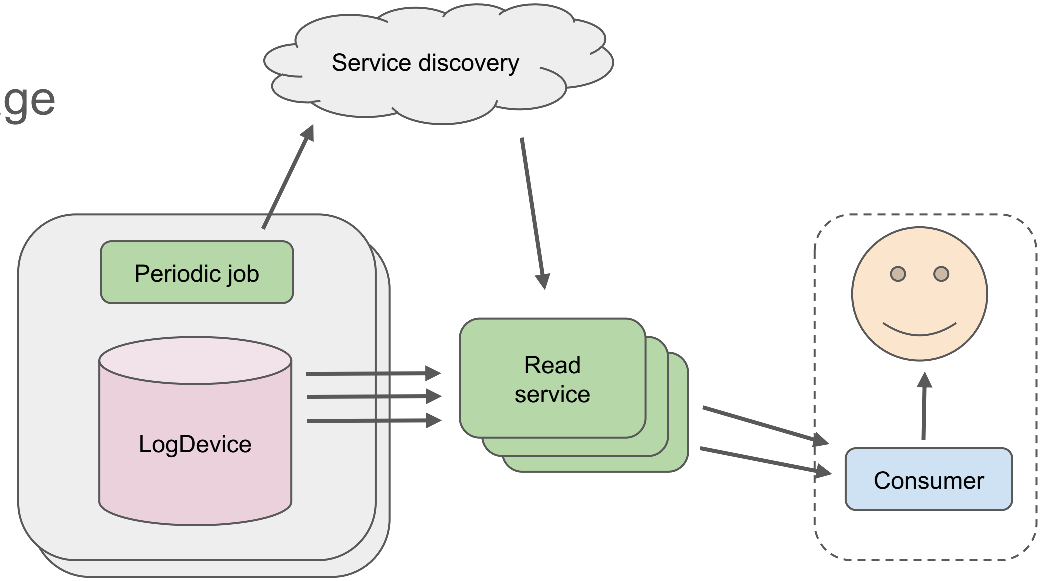
Single logical copy for each message

What if cluster unavailable:

- Data unavailable

Options

- Accept loss and carry on
- Wait
- Abort



"Clean" layering minimizes complexity yet is prone to data loss

(Rough) Ordering

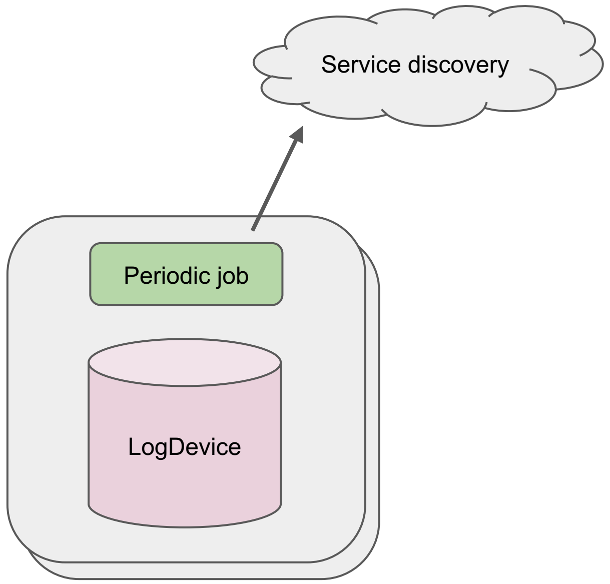
Storage

Each LogDevice cluster has a set of configured logs

- Each log holds data for only one category
- One category can have multiple logs
- Logs have a maximum throughput

Periodic job responsibilities

- Publish which categories have data
- Split logs if they get hot



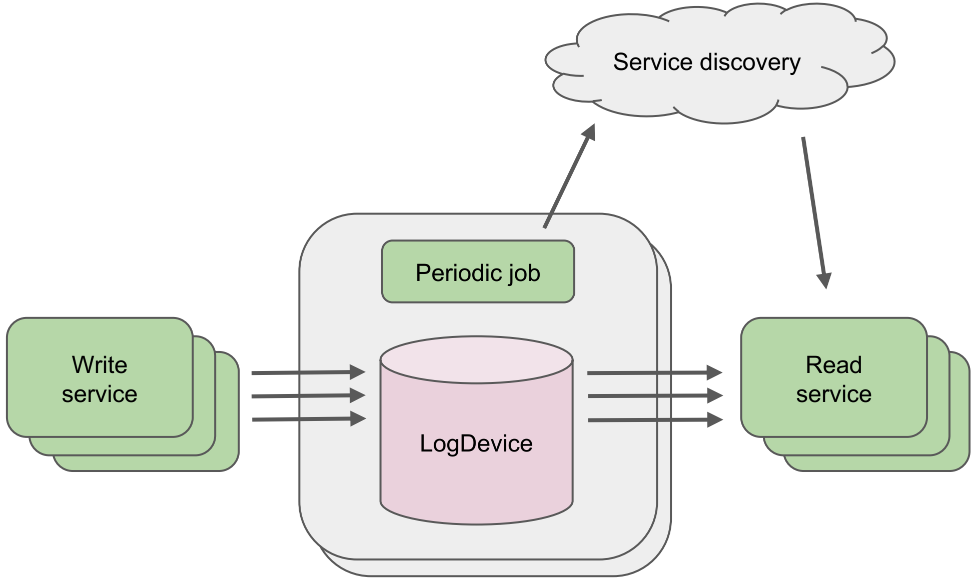
Storage

When Write Service sees (cat, msg)

- Pick a cluster
- Pick any log for cat
- Append msg

When read service sees (cat, time)

- Lookup clusters with data for cat
- Lookup time in all relevant logs
- Merge streams for logs into single output



Properties

Traffic to a single category scales horizontally

Consecutive writes can end up in entirely different clusters

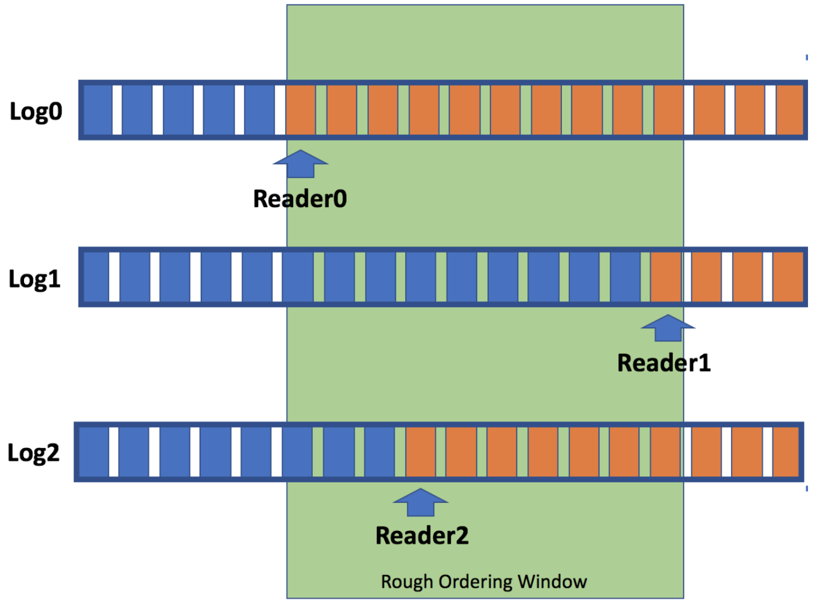
Top priorities

- Scalability
- (Write) availability

Lost in the process

- Ordering guarantees
- Repeatable reads

Rough ordering



Reduce blast radius of stragglers

Relaxed semantics in favor of read availability

There is no free lunch

Inherent trade-off between **scale**, operational **complexity**, and **semantics**

- Often, the semantics are held constant (“no loss”, “strict ordering”)
- In Scribe’s case, scale is imposed by company growth
- Relaxing semantics as a tool to manage complexity
- Users can still build (more) reliable apps over Scribe (at an extra cost)

When relaxing semantics, let users decide

Further information

[2019] Facebook eng blog post

engineering.fb.com/data-infrastructure/scribe

[2019] Tech talk Systems@Scale NYC

facebook.com/atscaleevents/videos/509450066277552

[2016] Realtime Data Processing at Facebook

research.fb.com/publications/realtime-data-processing-at-facebook

Questions?

Background

Scribe has been around for 10+ years

Initial purpose was to batch and store logs

Purpose evolved a lot over the years

Rearchitected multiple times to cope with scale

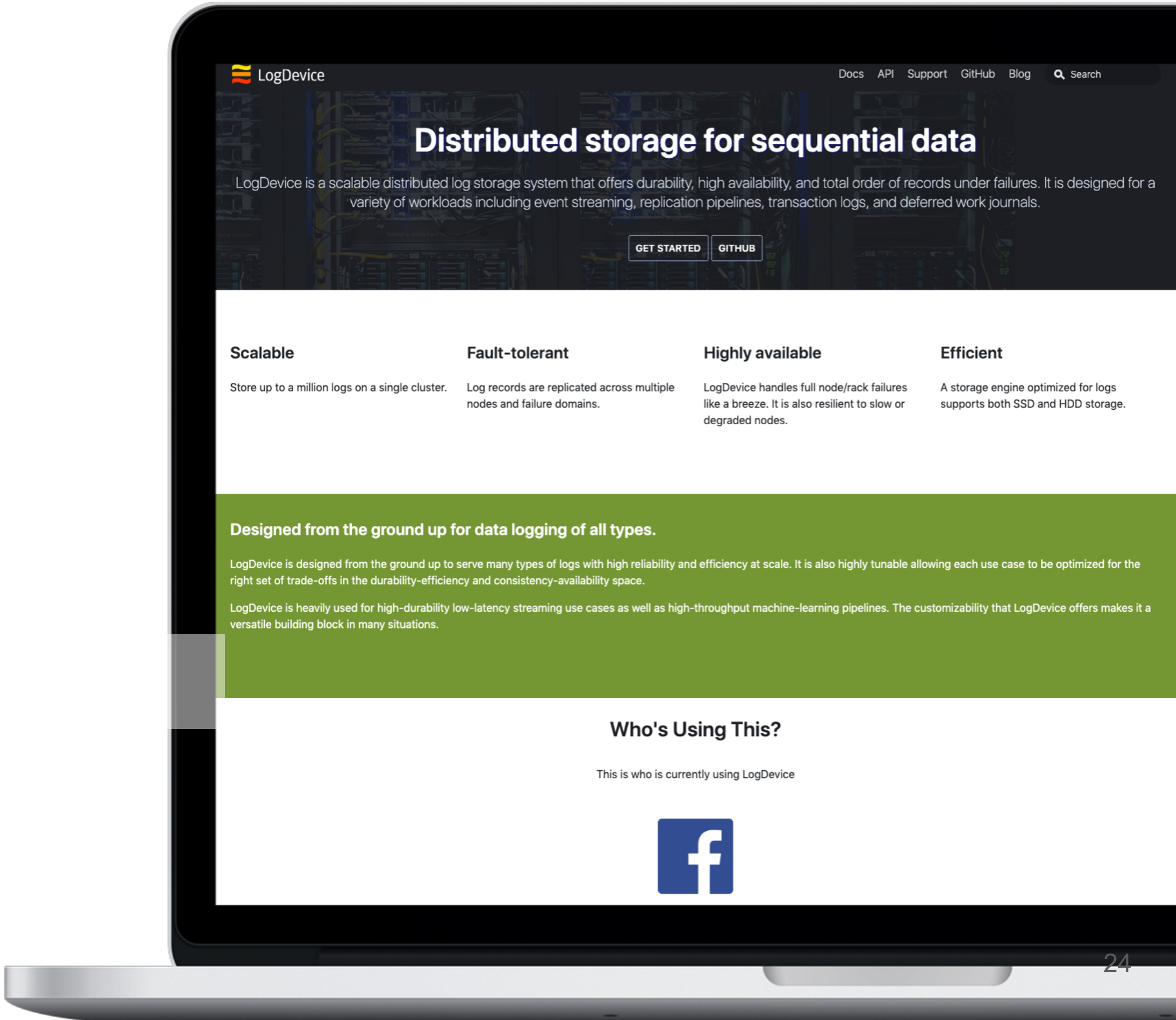
LogDevice

Distributed storage

Log as a primitive

Ideal for streams of data

logdevice.io



Distributed storage for sequential data

LogDevice is a scalable distributed log storage system that offers durability, high availability, and total order of records under failures. It is designed for a variety of workloads including event streaming, replication pipelines, transaction logs, and deferred work journals.

Scalable

Store up to a million logs on a single cluster.

Fault-tolerant

Log records are replicated across multiple nodes and failure domains.

Highly available

LogDevice handles full node/rack failures like a breeze. It is also resilient to slow or degraded nodes.

Efficient

A storage engine optimized for logs supports both SSD and HDD storage.

Designed from the ground up for data logging of all types.

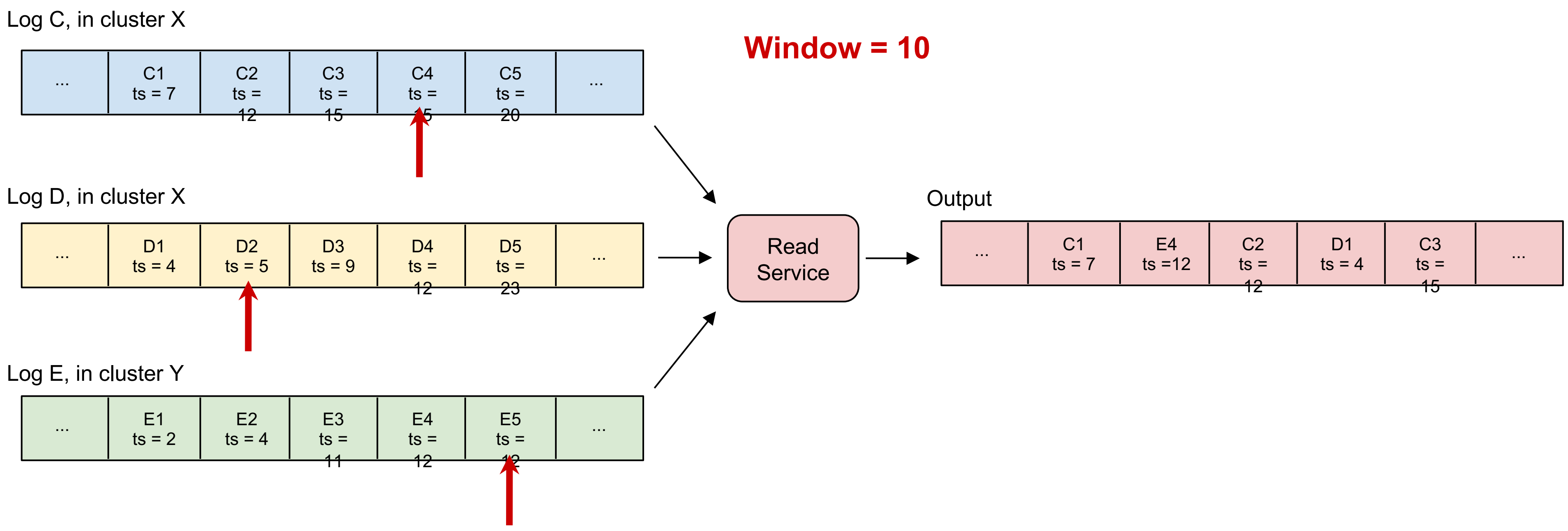
LogDevice is designed from the ground up to serve many types of logs with high reliability and efficiency at scale. It is also highly tunable allowing each use case to be optimized for the right set of trade-offs in the durability-efficiency and consistency-availability space.

LogDevice is heavily used for high-durability low-latency streaming use cases as well as high-throughput machine-learning pipelines. The customizability that LogDevice offers makes it a versatile building block in many situations.

Who's Using This?

This is who is currently using LogDevice

Rough ordering



Reduce blast radius of stragglers

Relaxed semantics in favor of read availability